Python: Opérations de comparaison

Les opérateurs de comparaison en Python (et dans de nombreux autres langages) vous permettent de comparer des valeurs pour évaluer des conditions et prendre des décisions dans votre code. Dans cet article, nous allons explorer les opérateurs de comparaison en Python, comment les utiliser et à quoi ils servent.

3GMS 5GMS 6GMS 3TTR 4GMS Découverte

Qu'est-ce qu'un Opérateur de Comparaison?

En programmation, un opérateur de comparaison est un **symbole** ou un mot-clé qui permet de **comparer deux valeurs ou expressions**. Le résultat de cette comparaison est une valeur booléenne, c'est-à-dire `True` (vrai) ou `False` (faux). Les opérateurs de comparaison vous permettent de répondre à des questions telles que "Est-ce que ces valeurs sont **égales** ?" ou "Est-ce que cette valeur est **plus grande** que celle-là ?".

En Python, voici les principaux opérateurs de comparaison :

- `==` : Égal à
- `!=` : Différent de
- `<` : Inférieur à
- `>` : Supérieur à
- `<=` : Inférieur ou égal à
- `>=` : Supérieur ou égal à
- `... <= x <=...` : intervalle

Exemples d'Utilisation

Voyons quelques exemples d'utilisation de ces opérateurs :

Égal à (`==`)

```
a = 5
b = 5
resultat = (a == b)  # Cela renverra True car a est égal à b.
print("Résultat:", resultat) # True
```

💡 L'opérateur de comparaison est `==` (2x le signe `=`) **afin de ne pas le confondre avec l'opérateur d'affectation**.
Taper ce code dans un éditeur:

```
1  a = 5
2  b = 15
3
4  print ( a == b)
5  print ( a = b)
```

Que se passe-t-il à la ligne 4 et à la ligne 5 ?

Différent de (!=)

```
1  x = 10
2  y = 20
3  resultat = (x != y)  # Cela renverra True car x est différent de y.
4  print("Résultat:", resultat) # True
```

Inférieur à (<)

```
1  age = 15
2  limite_age = 18
3  resultat = (age < limite_age)  # Cela renverra True car age est inférieur à limite_age.
4  print("Résultat:", resultat) # True
```

Supérieur à (>)

```
1  note = 85
2  note_passage = 60
3  resultat = (note > note_passage)  # Cela renverra True car note est supérieur à note_passage.
4  print("Résultat:", resultat) # True
```

Inférieur ou égal à (<=)

```
1  temps = 25
2  limite_temps = 30
3  resultat = (temps <= limite_temps)  # Cela renverra True car temps est inférieur ou égal à lim
4  print("Résultat:", resultat) # True
```

Supérieur ou égal à (>=)

```python
nombre1 = 50
nombre2 = 50
resultat = (nombre1 >= nombre2)  # Cela renverra True car nombre1 est supérieur ou égal à nomb
print("Résultat:", resultat) # True
```

Intervalles

En Python, il est possible de vérifier si une valeur se trouvent entre deux bornes, comme dans l'expression suivante :

```python
-50 <= temperature <= 50
```

Cette syntaxe permet de vérifier en une seule étape si une valeur se trouve dans un certain intervalle. Cela revient à combiner deux comparaisons logiques à l'aide de l'opérateur logique `and`. Concrètement, l'expression ci-dessus est équivalente à :

```python
-50 <= temperature and temperature <= 50
```

Explication technique :

1. **Lecture fluide** : L'utilisation d'une double inégalité améliore la lisibilité du code, car elle s'apparente à une notation mathématique couramment utilisée.
2. **Évaluation de gauche à droite** : Python évalue les expressions de gauche à droite. D'abord, il vérifie si `-50 <= temperature`. Si cette condition est vraie, il continue avec `temperature <= 50`. Si l'une des deux conditions échoue, l'évaluation s'arrête immédiatement.
3. **Type de valeur retournée** : Le résultat de la comparaison est un booléen (`True` ou `False`).

Exemple concret :

Voici un exemple qui vérifie si une température est dans une plage réaliste pour des mesures météorologiques :

```python
temperature = 25

if -50 <= temperature <= 50:
    print("La température est dans la plage acceptable.")
else:
    print("La température est hors plage !")
```

Dans cet exemple, la valeur `25` répond à la condition `-50 <= temperature <= 50`, donc le programme affichera :

```
La température est dans la plage acceptable.
```

Cas d'utilisation :

- Vérification de plages de valeurs numériques (par exemple, température, âge, scores, etc.).
- Validation de données dans des intervalles bien définis.
- Simplification des conditions multiples pour améliorer la lisibilité du code.

Bonnes pratiques :

- Préférez cette notation compacte lorsque vous travaillez avec des comparaisons d'intervalles.

- Assurez-vous que cette syntaxe est claire pour votre public cible ou documentez-la si nécessaire pour éviter toute confusion.

Combinaison d'Opérateurs de Comparaison

Vous pouvez également combiner plusieurs opérateurs de comparaison pour créer des expressions plus complexes. Par exemple, vous pouvez utiliser les opérateurs `and` et `or` pour combiner des conditions.

```
age = 25
sexe = "Femme"
resultat = (age >= 18) and (sexe == "Femme")  # Cela renverra True si l'âge est supérieur ou é
print("Résultat:", resultat) # True
```

Lire l'article complet

Pratique

Voici une série de comparaisons. Essaie d'abord de deviner le résultat de chacune d'entre elles. Tape ensuite ce code dans un éditeur pour vérifier tes hypothèses.

```
a = 5
b = 7
print(a == b)
print(a != b)
print(a > b)
print(a < b)
print(a >= b)
print(a <= b)
```

Bonus

Quel est le résultat du code suivant?

```
a = 25
b = "25"
print (a == b)
```

La réponse se trouve ici

Conclusion

Les opérateurs de comparaison sont des outils puissants pour évaluer des conditions et prendre des décisions dans vos programmes Python. Ils vous permettent de comparer des valeurs de manière simple et efficace. En

comprenant comment utiliser ces opérateurs, vous serez en mesure d'écrire un code plus précis et plus adapté à vos besoins. N'hésitez pas à les utiliser pour résoudre des problèmes et créer des applications plus sophistiquées.

Vidéo à la une à regarder sur Youtube: http://youtu.be/WNJRn4LbxXs